

Recuperatorio del Segundo Parcial

Primer cuatrimestre de 2019

- Apagar los celulares.
- Hacer cada ejercicio en hojas separadas.
- Poner nombre, número de orden y número de página en cada ejercicio.
- Justificar todas las respuestas.
- El examen es a libro abierto.
- Se aprueba con 65 puntos.

1. (33 pts) Dada la siguiente gramática:

$G_1 = \langle \{E, K, V\}, \{ (,), =, id \}, E, P \rangle$, con P :

$$E \rightarrow (K V)$$

$$K \rightarrow id = \mid \lambda$$

$$V \rightarrow id \mid E$$

Determinar a cuáles de las siguientes clases de gramáticas pertenece G_1 :
LR(0), SLR, LALR, LR(1).

2. (33 pts) La siguiente gramática genera un subconjunto de los constructores de tablas del lenguaje Lua:

$G_2 = \langle \{C, L, F, E\}, \{ \{, \}, [,], ,, =, num, string, id \}, C, P \rangle$, con P :

$$C \rightarrow \{ L \} \mid \{ \}$$

$$L \rightarrow L , F \mid F$$

$$F \rightarrow [E] = E \mid id = E \mid E$$

$$E \rightarrow num \mid string \mid id \mid C$$

Dar una gramática extendida que genere $L(G_2)$ y sea ELL(1).

3. (34 pts) Dar una gramática de atributos que acepte cadenas del lenguaje:

$L_3 = \{ a^k b \omega \mid k \geq 1 \wedge \omega \in \{a, b\}^* \wedge \text{ toda subcadena maximal de } a \text{ es consecutiva de } \omega \text{ tiene longitud mayor que } k \}$

Ejemplos: $aabbbaaaabaaab \in L_3$, $aabgbaaaaa \notin L_3$.

1	2	3	4
3	33	39	100

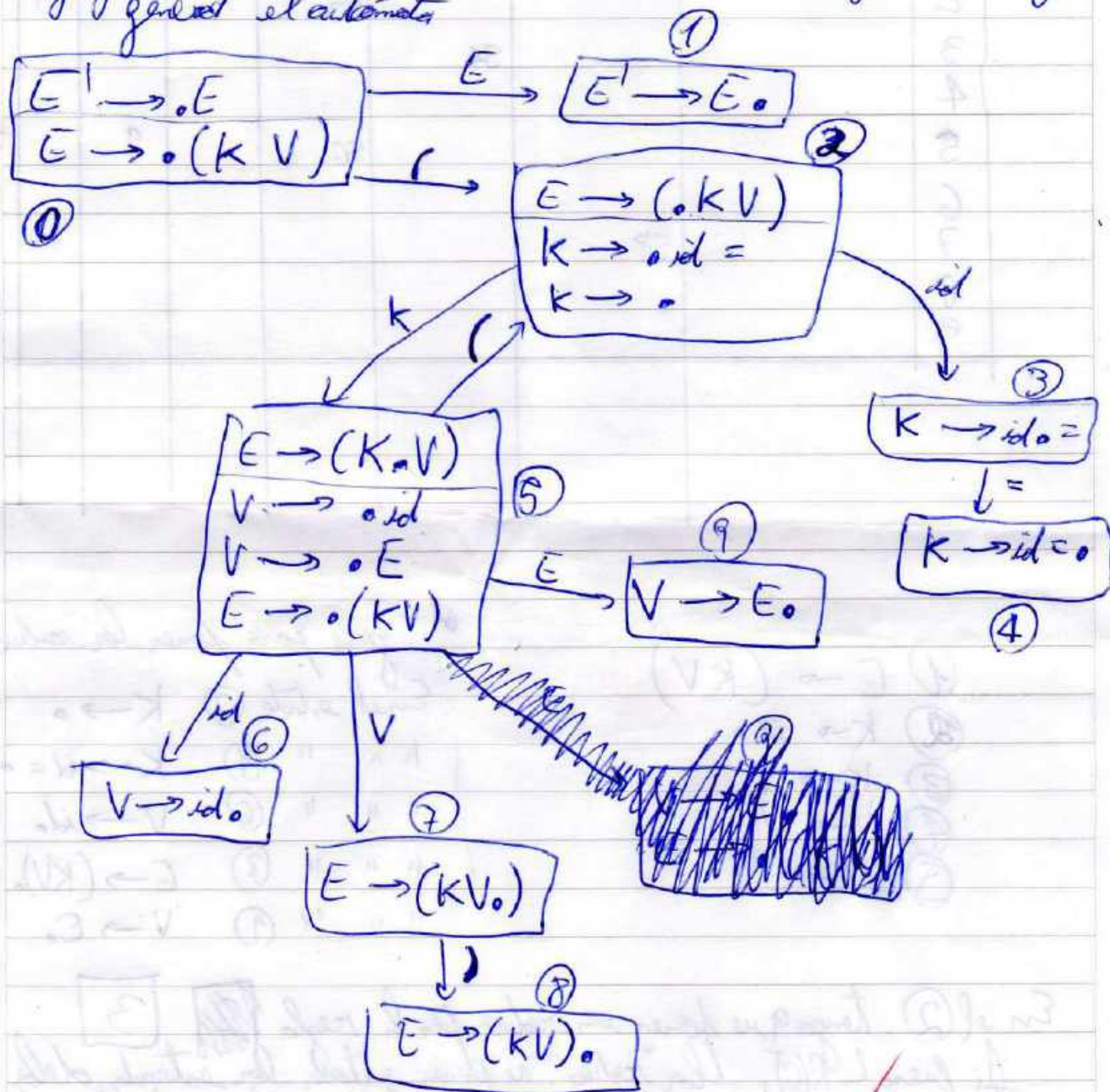


WORD 4

HOJA 1

1) Chequea si es LR(0) / SLR

Agrega la producción $E' \rightarrow E$ a la gramática y genera el automata



$E \rightarrow (KV)$
 $K \rightarrow id =$
 $K \rightarrow \lambda$
 $V \rightarrow id$
 $V \rightarrow E$

Armo la tabla

	()	=	id	\$	E	K	V
0	S2					1		
1	#				accept			
2				S3			5	
3			S4					
4								
5				S6		9		7
6								
7		S8						
8								
9								

① $E \rightarrow (KV)$

② $K \rightarrow id =$

③ $K \rightarrow \lambda$

④ $V \rightarrow id$

⑤ $V \rightarrow E$

Guía para poner las reglas

En el estado ② $K \rightarrow \lambda$

" " " ④ $K \rightarrow id =$

" " " ⑥ $V \rightarrow id$

" " " ③ $E \rightarrow (KV)$

" " " ① $V \rightarrow E$

En el ② tengo que poner un reduce por la regla 3
 Si fuera LR(0), las "reduc" se ponen antes en la entrada de los
 terminales, lo que produciría un conflicto en la entrada "id"
 ya que esto tendría S3/r3 \Rightarrow no es LR(0) ✓

Si fuera SLR las "reduc" se ponen solo en la "siguiente" del
 no terminal de la producción correspondiente (en este caso K,
 porque la producción es $K \rightarrow \lambda$)

#ORD 4

HOJA 2

Calcular siguiente(K)

K aparece a la derecha de la prod. $E \rightarrow (KV)$ donde
Como V no es anulable entonces $\text{siguiente}(K) = \text{primero}(V)$

$$\text{primero}(V) = \{id, (\}$$

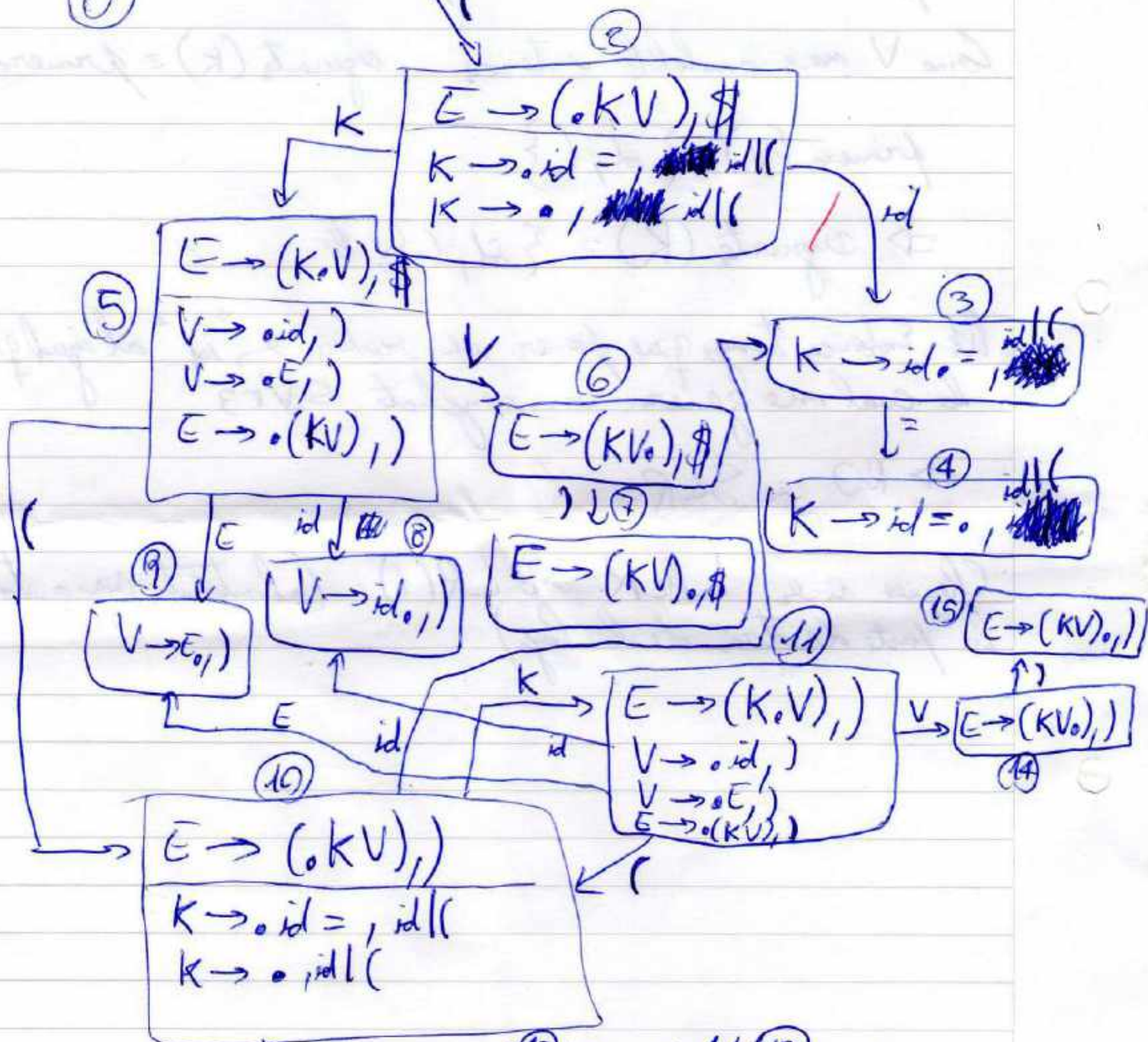
$$\Rightarrow \text{siguiente}(K) = \{id, (\}$$

Entonces tengo que poner un $\$$ en "id" al igual que antes,
lo cual me genera un conflicto S3/R3

\Rightarrow NO es SLR

Cheques si es LALR o LR(1) (el autómata va a estar en la parte de atrás de la hoja)

Extendiendo la gramática con $E' \rightarrow E$



~~Items 12 and 13 are crossed out.~~

- ① $E \rightarrow (KV)$
- ② $K \rightarrow id =$
- ③ $K \rightarrow \lambda$
- ④ $V \rightarrow id$
- ⑤ $V \rightarrow E$

calcula lookahead states

① $\rightarrow \text{primeros}(KV\#\$) = \{id, (, \}$

② $\rightarrow \text{primeros}(V)\# = \{id, (, \}$

⑤ $\rightarrow \text{primeros}()\# = \{ \}$ ⑩ $\rightarrow \text{prim}(V)) = \{id, (, \}$

11) $\text{prim}(()) = \{ \}$
 $\text{prim}() = \{ \}$

Armo la tabla LR(1)

	()	=	id	\$	E	K	V
0	s2					1		
1					accept			
2				s3			5	
3			s4					
4								
5	s10			s8		9		6
6		s7						
7								
8								
9								
10				s3			11	
11	s10			s8		9		14
14		s15				9		
15								

Para las reducciones

- 1) $E \rightarrow (KV)$
- 2) $K \rightarrow id$
- 3) $K \rightarrow \lambda$
- 4) $V \rightarrow id$
- 5) $V \rightarrow E$

En el estado 2 tengo que reducir a 3
 en la lookahead "id" "(" lo cual seguramente
 generaría un conflicto en la tabla en la posición
 (2, id) s3/r3 ✓

→ no es LR(1) → no es LALR → no es SLR → no es LR(0) ✓

→ G no pertenece a ninguna de estas clases de gramáticas

$$2) C \rightarrow \{L\} | \{\}$$

$$L \rightarrow L, F | F$$

$$F \rightarrow [E] = E | id = E | E$$

$$E \rightarrow num | string | id | C$$

quiero que la gramática extendida sea LL(1)

$L \rightarrow L, F | F$ tiene recursión a izquierda

la cambio por

$$L \rightarrow F X$$

$$X \rightarrow , F X | \lambda$$

Entonces tengo

$$C \rightarrow \{L\} | \{\}$$

$$L \rightarrow F X$$

$$X \rightarrow , F X | \lambda$$

$$F \rightarrow [E] = E | id = E | E$$

$$E \rightarrow num | string | id | C$$

además, factorizo $C \rightarrow \{L\} | \{\}$

$$C \rightarrow \{Y$$

$$Y \rightarrow L\} | \}$$

y me queda la siguiente gramática

$C \rightarrow \{Y$
 $Y \rightarrow L\} \}$
 $L \rightarrow FX$
 $X \rightarrow ,FX | \lambda$
 $F \rightarrow [E] = E | id = E | E$
 $E \rightarrow num | string | id | C$

$C \rightarrow \{Y$
 $Y \rightarrow L\}$
 $Y \rightarrow \}$
 $L \rightarrow FX$
 $X \rightarrow ,FX$
 $X \rightarrow \lambda$
 $F \rightarrow [E] = E$
 $F \rightarrow id = E$
 $F \rightarrow E$
 $E \rightarrow num$
 $E \rightarrow string$
 $E \rightarrow id$
 $E \rightarrow C$

Checkear si es LL(1)

Para eso tiene que valer

$$\begin{aligned}
 SD(Y \rightarrow L\}) \cap SD(Y \rightarrow \}) &= \emptyset \\
 SD(X \rightarrow ,FX) \cap SD(X \rightarrow \lambda) &= \emptyset \\
 SD(F \rightarrow [E] = E) \cap SD(F \rightarrow id = E) &= \emptyset = SD(F \rightarrow [E] = E) \cap SD(F \rightarrow E) \\
 SD(F \rightarrow id = E) \cap SD(F \rightarrow E) &= \emptyset \\
 SD(E \rightarrow num) \cap SD(E \rightarrow string) &= \emptyset = SD(E \rightarrow num) \cap SD(E \rightarrow id) \\
 SD(E \rightarrow num) \cap SD(E \rightarrow C) &= \emptyset = SD(E \rightarrow string) \cap SD(E \rightarrow id) \\
 SD(E \rightarrow string) \cap SD(E \rightarrow C) &= \emptyset = SD(E \rightarrow id) \cap SD(E \rightarrow C)
 \end{aligned}$$

Calcular primero en $X \rightarrow ,FX | \lambda$

$$\begin{aligned}
 SD(X \rightarrow ,FX) &= primeros(,FX) = \{, \} \\
 SD(X \rightarrow \lambda) &= siguientes(X) = \{ ' \} \\
 \cap &= \emptyset
 \end{aligned}$$

de la producción
 $L \rightarrow FX$

debe que $sig(L) \subseteq sig(X)$

[Signature]

de $X \rightarrow ,FX$ no debe tener información extra sobre $sig(X)$

$$\text{siguientes}(L) = \{ \{ \} \}$$

$$\text{de } Y \rightarrow L \}$$

de tal modo que $\{ \in \text{siguientes}(L)$

la intersección
es vacía

Ahora chequea $Y \rightarrow L \{ \}$

$$SD(Y \rightarrow L \{ \}) = \text{primeros}(L \{ \}) = \{ '[', 'id', 'num', 'string', '\{ \}' \}$$

$$SD(Y \rightarrow \{ \}) = \text{primeros}(\{ \}) = \{ '\{ \}' \}$$

$$\text{primeros}(L \{ \}) = \text{primeros}(L) = \{ '[', 'id', 'num', 'string', '\{ \}' \}$$

↓
L no es nullable

Chequea $F \rightarrow [E] = E \mid id = E \mid E$

$$SD(F \rightarrow [E] = E) = \text{primeros}([E] = E) = \{ '[\}$$

$$SD(F \rightarrow id = E) = \text{primeros}(id = E) = \{ 'id' \}$$

$$SD(F \rightarrow E) = \text{primeros}(E) = \{ num, id, \dots \}$$

→ la intersección
es NO vacía

reescriba la gramática (el truco es llevar 'id' a F para hacerla desaparecer de E)

$$\begin{aligned} C &\rightarrow \{ Y \\ Y &\rightarrow L \{ \} \} \\ L &\rightarrow F X \\ X &\rightarrow , F X \mid \lambda \end{aligned}$$

$$\begin{aligned} F &\rightarrow [R] = R \mid * V \mid id F' \\ F' &\rightarrow = F'' \mid \lambda \\ F'' &\rightarrow W \mid id \quad \text{C} \\ W &\rightarrow num \mid string \mid C \\ R &\rightarrow num \mid string \mid id \mid C \end{aligned}$$

✓

$$SD(Y \rightarrow L) = \text{prim}(L) = \{ '[', 'num', 'string', '\{', 'id' \}$$

$$SD(Y \rightarrow \{) = \{ '\{ ' \} \} \rightarrow \cap = \emptyset$$

$$SD(X \rightarrow, FX) = \{ ', ' \}$$

$$SD(X \rightarrow \lambda) = \text{rig}(X) = \text{rig}(L) = \{ '[', 'num', 'string', '\{', 'id' \} \} \rightarrow \cap = \emptyset$$

$$SD(F \rightarrow [R] = R) = \{ '[' \}$$

$$SD(F \rightarrow W) = \{ 'num', 'string', '\{ ' \}$$

$$SD(F \rightarrow id(F)) = \{ 'id' \}$$

$\cap = \emptyset$ *intersección*
2 a 2
vacía

$$SD(F' \rightarrow = F'') = \{ '=' \}$$

$$SD(F' \rightarrow \lambda) = \text{rig}(F') = \text{rig}(F) = \{ ', ' \} \cup \text{rig}(X) \cup \text{rig}(L)$$

$$\cap = \emptyset = \{ ', ' , '\{ ' \}$$

$$SD(F'' \rightarrow W) = \{ 'num', 'string', '\{ ' \}$$

$$SD(F'' \rightarrow id) = \{ 'id' \} \rightarrow \cap = \emptyset$$

$$SD(W \rightarrow num) = \{ 'num' \}$$

$$SD(W \rightarrow string) = \{ 'string' \}$$

$$SD(W \rightarrow C) = \{ '\{ ' \}$$

$\cap = \emptyset$ *intersección*
2 a 2

$$SD(R \rightarrow num) = \{ 'num' \}$$

$$SD(R \rightarrow string) = \{ 'string' \}$$

$$SD(R \rightarrow id) = \{ 'id' \}$$

$$SD(R \rightarrow C) = \{ '\{ ' \}$$

$\cap = \emptyset$ *intersección*
2 a 2

∴ la gramática es LL(1)

$$\tilde{G}_2 = \langle \{C, Y, L, X, F, F', F'', W, R\}, \\ \{ \{ ' ' \}, [' '], ' ', '=', 'num', 'string', 'id' \}, \\ C, \tilde{P} \rangle$$

con \tilde{P} :

$$\begin{aligned} C &\rightarrow \{Y \\ Y &\rightarrow L \{ \} \} \\ L &\rightarrow FX \\ X &\rightarrow , FX | \lambda \\ F &\rightarrow [R] = R | W | id F' \\ F' &\rightarrow = F'' | \lambda \\ F'' &\rightarrow W | id \\ W &\rightarrow num | string | C \\ R &\rightarrow num | string | id | C \end{aligned}$$

Haciendo un abuso de notación se pueden interpretar las producciones $X \rightarrow \alpha_1 | \dots | \alpha_m$ como una gramática extendida con $\alpha_1 | \dots | \alpha_m$ una expresión regular, \tilde{G}_2 es una gramática extendida, y como $EL(1)$, o $EL(1)$.

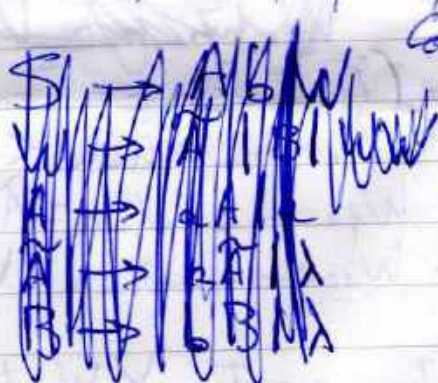
Nota: para solucionar el problema de 'id' en la producción $F \rightarrow$, lleve todo primero a una forma donde se podía factorizar id. Es decir ahí sale F' . Luego ve que el '=' se podía factorizar también (de ahí sale F'') y era algo como resultado a

3) $a^k b w$
 $\{a, b\}^*$

Es necesario que controle la cantidad de a -s que vienen al principio

Idea: ~~para~~ por un lado tome la cantidad de a -s del principio, y por el otro me quede con el mínimo de la longitud de las subcadenas maximales de a -s consecutivos en w . Así se puede comparar y aceptar o descartar una cadena según corresponda.

$G = \langle \{S, W, A, \tilde{A}, B\}, \{a, b\}, S, P \rangle$



Con P:

- $S \rightarrow AbW$
- $W \rightarrow \tilde{A}W | BW | \lambda$
- $A \rightarrow aA | a$
- $\tilde{A} \rightarrow a\tilde{A} | \lambda$
- $B \rightarrow bB | \lambda$

• $B \rightarrow \lambda$ $\{ B.\text{primera} = '1', B.\text{min-cont-}a = 0, B.\text{tiene-}a = \text{false} \}$

• $B_1 \rightarrow bB_2$ $\{ B_1.\text{primera} = 'b', B_1.\text{min-cont-}a = B_2.\text{min-cont-}a, B_1.\text{tiene-}a = B_2.\text{tiene-}a \}$

• $\tilde{A} \rightarrow \lambda$ $\{ \tilde{A}.\text{primera} = '1', \tilde{A}.\text{min-cont-}a = 0, \tilde{A}.\text{tiene-}a = \text{false} \}$

$$\hat{A}_1 \rightarrow a \hat{A}_2 \quad \{ \hat{A}_1.\text{primera} = 'a'; \hat{A}_1.\text{min-cont-a} = 1 + \hat{A}_2.\text{min-cont-a}; \hat{A}_1.\text{true-a} = \text{true} \}$$

$$A \rightarrow a \quad \{ A.h = 1 \}$$

$$A_1 \rightarrow a A_2 \quad \{ A_1.h = 1 + A_2.h \}$$

$$W \rightarrow \lambda \quad \{ W.\text{primera} = '\lambda'; W.\text{min-cont-a} = 0; W.\text{true-a} = \text{false} \}$$

$$W_1 \rightarrow \hat{A} W_2 \quad \{ \text{condition } (W_2.\text{primera} == 'a' \vee W_2.\text{primera} == 'b') \}$$

$$W_1.\text{primera} = 'a';$$

~~$$W_1.\text{min-cont-a} = \begin{cases} \hat{A}.\text{min-cont-a} & \text{if } \hat{A}.\text{true-a} == \text{true} \\ W_2.\text{min-cont-a} & \text{if } \hat{A}.\text{true-a} == \text{false} \end{cases}$$~~

$$W_1.\text{min-cont-a} = \begin{cases} \hat{A}.\text{min-cont-a} & \text{if } (\hat{A}.\text{true-a} == \text{true} \wedge \hat{A}.\text{min-cont-a} < W_2.\text{min-cont-a}) \\ W_2.\text{min-cont-a} & \text{if } (\hat{A}.\text{true-a} == \text{false} \vee \hat{A}.\text{min-cont-a} > W_2.\text{min-cont-a}) \end{cases}$$

$$W_1.\text{true-a} = \begin{cases} \text{true} & \text{if } (\hat{A}.\text{true-a} == \text{true} \vee W_2.\text{true-a} == \text{true}) \\ \text{false} & \text{if } (\hat{A}.\text{true-a} == \text{false} \wedge W_2.\text{true-a} == \text{false}) \end{cases}$$

Acá debería ser 'a'

$$W_1 \rightarrow B W_2 \quad \{ \text{condition } (W_2.\text{primera} == 'a' \vee W_2.\text{primera} == 'b') \}$$

$$W_1.\text{primera} = 'b';$$

~~$$W_1.\text{true-a} = \begin{cases} \text{true} & \text{if } (B.\text{true-a} == \text{true} \vee W_2.\text{true-a} == \text{true}) \\ \text{false} & \text{if } (B.\text{true-a} == \text{false} \wedge W_2.\text{true-a} == \text{false}) \end{cases}$$~~

~~$$W_1.\text{min-cont-a} = \begin{cases} B.\text{min-cont-a} & \text{if } B.\text{true-a} == \text{true} \\ W_2.\text{min-cont-a} & \text{if } B.\text{true-a} == \text{false} \end{cases}$$~~

$$W_1.\text{true-a} = W_2.\text{true-a}$$

$$W_1.\text{min-cont-a} = W_2.\text{min-cont-a}$$

$S \rightarrow ABW$ { condition ($W.tiene_a == false$
 $W.min_cont_a > A.k$) }

$S.tiene_a = true$
 $S.primer_a = 'a'$
 $S.min_cont_a = A.k$ }

Si W no tiene ningunos 'a' entonces tengo que aceptar la cadena.
Si tiene algunos 'a' entonces chequeo que la longitud de la máxima
subcadena maximal de 'a' consecutivos en W sea mayor que
K, con lo cual todo lo demás cumplirá \Rightarrow acepto la cadena.

	atributo	tipo	S/H
B	primera	char	S
	min-cont-a	int	S
	tiene-a	bool	S
A	primera	char	S
	min-cont-a	int	S
	tiene-a	bool	S
A	k	int	S
W	primera	char	S
	min-cont-a	int	S
	tiene-a	bool	S
S	primera	char	S
	min-cont-a	int	S
	tiene-a	bool	S